Enhancing Story Generation with the Semantic Web

Eric LaBouve
California Polytechnic State University
San Luis Obispo, California
elabouve@calpoly.edu

Erik Miller
California Polytechnic State University
San Luis Obispo, California
emille26@calpoly.edu

Foaad Khosmood
California Polytechnic State University
San Luis Obispo, California
foaad@calpoly.edu

ABSTRACT

In story or character driven games, in-game stories are usually manually authored in advance. As the complexity of interactions in games increases, the quantity of hand-crafted text typically follows. Designing stories and composing content by hand is a laborious and time consuming process that if automated, would speed up game production and lower development costs. In this paper, we present a mixed initiative tool to help generalize and enhance context free grammars (CFGs) for story generation. The tool is designed to take as input a story generating grammar in addition to generic keywords for people, places and other various metrics in order to control the output text. The tool is knowledgeable about a wide array of topics because it leverages the *Semantic Web* in order to extrapolate more details and related information from the user supplied content. As a result, generated text will contain genuinely new information, descriptions of characters and locations that were never written by the author. Although the general structure of a story or dialogue is somewhat fixed by the nature of grammar rules, the resulting text can be geared towards a variety of user inputs and can include details that may surprise designers. The tool is evaluated by a group of 15 individuals in a user study to gauge the practicality of using Semantic Web technologies for procedural text generation. The study concludes that using the Semantic Web is an effective aid for grammar based text generation. We discuss our system, the user study and share thoughts on future work.

CCS CONCEPTS

• **Information systems → Web crawling**; **Information retrieval query processing**; *Web searching and information discovery*; • **Computing methodologies → Information extraction**; *Symbolic and algebraic algorithms*; • **Human-centered computing →** Human computer interaction (HCI);

KEYWORDS

Context Free Grammar, Procedural Content Generation, Semantic Web, Sparql, Resource Description Framework, DBPedia

ACM Reference Format:
Eric LaBouve, Erik Miller, and Foaad Khosmood. 2019. Enhancing Story Generation with the Semantic Web. In *The Fourteenth International Conference on the Foundations of Digital Games (FDG '19), August 26–30, 2019, San Luis Obispo, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3337722.3337742

1 INTRODUCTION

One of the most explored areas of procedural text generation is the use of context free grammars. With enough effort, an author can design a sophisticated grammar that can generate text output which can dramatically vary in size and content when grammar rules are randomly expanded. However, we identify two major limitations to using context free grammars for procedural text generation. First, writing grammar rules is a time intensive procedure. This is a consequence of the combinatorial explosion of paths that can be expanded when there are many rules. As a result, it takes a long time to incorporate many story details. Second, once a set of grammar rules is written, the rules cannot easily be reused in different contexts because the logic and output of the grammar is defined ahead of time [24]. As a result, authors may resort to writing different grammars for stories that share similar structure or dynamically building grammars using general purpose programming languages. For example, stories often introduce new characters with relevant background information, such as a character's place of birth, occupation, family members, etc. If the author wants to introduce another character in a similar manner to the first character, the author would need to derive new character descriptions and compose a second grammar. These two grammars may contain structural similarities that can be abstracted into a single set of grammar rules. Furthermore, coming up with accurate and unique character descriptions is a time consuming exercise that is of interest for automation in our research.

Researching ways to write grammar rules that can generalize to different contexts and produce descriptions that were not explicitly written by the author ahead of time is an interesting and important problem in procedural text generation because it has applications for story telling and other text generation systems. Additionally, it is also important to empower authors with the ability to control the level of story descriptions and story length in order to satisfy author-specific use cases. Such an adaptive grammar could result in a highly immersive experience for the reader and allow authors to devote their attention to other important activities. Unfortunately, building a system that generates believably human-like stories and dialogues is a Turing-complete problem.

We present a tool that uses Semantic Web technologies, which are fundamental tools used by search engines for query answering. Our tool is knowledgeable on a huge set of topics, pertaining to people and locations, by leveraging information scraped from Wikipedia. In order to allow easy access to this data, the structured content from Wikipedia has been extracted into an ontology database, composed of many RDF triples, and is sponsored by DBPedia[1]. Using a query language called Sparql, we can query for metadata and incorporate this information into our generated stories.

According to a survey by Kybartas and Bidarra called *A Survey on Story Generation Techniques for Authoring Computational Narratives* [14], there exists a "burden" on the author to create *space*, which relates to everything that exists abstractly in the narrative. Through the power of Sparql and large RDF data sets, we hypothesize that Semantic Web technologies can be used to generate more in-depth text stories in order to help alleviate this burden of creating *space* in narratives. Our unique contribution is a mixed initiative tool that empowers authors with more control over rule expansions, streamlines the generation of intimate story details, and a grammar that can be generalized to many use cases. We evaluate our tool by conducting a user study with 15 individuals. After interacting with our tool in a variety of different use cases, the participants conclude that using Semantic Web technologies is an effective way to generate more in-depth text stories and there are many possibilities for future research.

The remainder of the paper is structured as follows. Section 2 covers background on Semantic Web technologies and related research on procedural content generation using Semantic Web technologies and for story and text generation. Section 3 explains the implementation of our tool, which will begin by discussing how Sparql is used to extrapolate semantic information from DBPedia, then discusses the rules of our grammar, and subsequently goes over the design of our mixed initiative tool. Afterwards in Section 4, we present an empirical user study of our tool. Lastly, Sections 5 and 6 will present concluding thoughts and future work.

2 BACKGROUND AND RELATED WORK

In this section, we provide a brief background on Semantic Web technologies and related work that uses Semantic Web technologies. Also, we review related work in procedural content generation for creating text and stories.

2.1 Semantic Web Technologies

The technology that allows Wikipedia to be represented as an ontology is a metadata model called RDF. RDF, which stands for Resource Description Framework, is a *Semantic Web* technology that was developed by an international standards organization called World Wide Web Consortium (W3C) to provide ontological relationships to elements on the web [18]. Many RDF data sets are hosted by DBPedia, which is self-defined on their website to be a "crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects". Metadata in the RDF model is expressed as triples: subject, predicate, and object, which are encoded as URI. For example, an RDF triple, where the subject is a URI pointing to the DBPedia entity for Barack Obama[2], the predicate is a URI pointing to the DBPedia property for spouse[3], and the object is a URI pointing to the DBPedia entity for Michelle Obama[4], indicates that Barack Obama and Michelle Obama are married to each other. RDF is the standard format across the web because its triple structure allows for easy joins between many RDF data sets. In order to search through an RDF database, a query language called Sparql is used. Sparql, which is a recursive acronym that stands for Sparql Protocol and RDF Query Language, is designed similarly to SQL and is easy to learn for developers that are already familiar with relational database query languages. An example Sparql query can be found in Figure 1, which is fully explained in Section 3.1.

Semantic Web technologies have found recent uses in procedural content generation. In the realm of game design, a notable example comes from the Data Adventures project [2]. The system described in this research creates game plots based on associations between Wikipedia articles. Using these associations, the game generates a sequence of clues, where the player's goal is to find the location of a non-playable character. Other researchers have taken procedural game development a step further by attempting to generate entire games via semantic information extracted from Semantic Web content [23]. On the other hand, in the space of text generation, Semantic Web technologies have been used to provide the intelligence for question-answering systems. In *Automatic Expressive Opinion Sentence Generation for Enjoyable Conversational Systems* [17] the authors build a text based conversation program that uses DBPedia and Sparql in order to generate interesting small-talk. The system is partly designed to convert a user's questions into Sparql queries and to leverage the massive amount of information that can be found in DBPedia to respond to users with human-like intelligence.

2.2 Procedural Content Generation for Text and Stories

Context free grammars first arose the 1960's [24] when Scheinberg laid out a formal definition based on previous work from Chomsky [11], stating that a "context free (CF) grammar to be a finite set G of "rewriting rules" $a \longrightarrow \psi$, where a is a single symbol and ψ is a finite string of symbols from a finite alphabet (vocabulary) V" [24]. This has proven to be a simple and useful mechanism for generating stories, as aspects of the story may be specified ahead of time, with some parts of the story chosen at random from the vocabulary to vary the resulting text.

Since the early days of context free grammars, there have been efforts to generalize grammar rules to fit custom stories [8] and impose "cause and effect" relationships by defining the relations between specific story elements and story events [20]. The main drawback of developing grammar based story generation systems is that the entire generative space must be defined and written ahead of time. Thus, generating very large stories becomes an involved process. However, there has been some work to help expand grammars to require less work from the human author. *Evolving stories: Grammar evolution for automatic plot generation* [7] discusses a human-in-the-loop method to take regular grammar and evolve it to create novel story points. While we are looking to create novelty by using outside information, this provides a useful perspective on

[1]https://wiki.dbpedia.org/
[2]http://dbpedia.org/page/Barack_Obama
[3]http://dbpedia.org/ontology/spouse
[4]http://dbpedia.org/page/Michelle_Obama

building a mixed initiative tool on top of a grammar based story generator.

Propp [21] introduced a way to describe character stereotypes and general plot themes in the form of "functions." Propp's ideas paved the way for rewrite rules, which consist of symbols and symbol expansions that are loosely based on grammars. Similar to how our system allows authors some control over the generated story, researchers have used rewrite rules to create template based interactive narrative systems to cooperatively work with computers to generate a plot [3]. Other mixed initiative tools have empowered authors to change specific plot events after they have already been generated [9]. As more research was conducted, rewrite rules were extended to more complex plot structures and applied to graphs in order to model higher dimensional relationships [15].

Context free grammars and rewrite rules laid the foundation for story generation using search based approaches. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* [25] outlines planning based approaches to generate game stories and NPC behavior. From the concepts described in this textbook and other research, many variations of text and plot generation using symbolic search have been published. For example, Cheong et al. demonstrates that if the components of a story are decomposed into *operators*, which consist of pre and post conditions, then generating a story can be represented as a search problem within a "state" or "plan" space [10]. The authors demonstrate that operators can be encoded as propositional logic in the form of either a STRIPS[12] or an ADL[13] style planning representation, two languages designed to formally describe the process of planning and enable automated decision making. Using propositional logic with pre and post conditions has also shown to be an effective way to influence story telling in other research projects [27]. Another algorithmic approach titled *Ice-Bound: Combining Richly-Realized Story with Expressive Gameplay* [22] generates stories using a middle ground algorithm that builds continuous, small, reversible choices. Their algorithm marries branching-path and simulations story telling approaches to allow players to participate in story generation. The authors note that stories generated using only branch-path algorithms are limited because they lead to a combinatorial explosion of possible plot lines and for interactive stories. We hope that some of the techniques introduced in our research can help alleviate this problem.

While search and planning has allowed authors to control the sequence of story events, some researchers have used planning to create surprise and drama. One research endeavor to stimulate surprise is through the use of foreshadowing and flashbacks [5]. Although this research takes advantage of story telling techniques to surprise the user, our research aims to surprise the user by incorporating intimate knowledge from DBPedia. Additionally, *(Re)telling Chess Stories as Game Content* [6] is a research project that uses chess game-play traces to generate dramatic stories with a high degrees of variation. In their paper, Buckthal and Khosmood hypothesize that since playing chess can invoke a sense of drama and entertainment, both between the two players and for spectators, then a story generated from a game of chess may also embody these elements. The downside to their system is that the variability of their stories is limited by the number of story skins their system has available to use because they are written manually (their examples include a Zombie Apocalypse and Romeo and Juliet). A tool that takes advantage of the Semantic Web may be able to help streamline skin generation by inferring details from other well known stories such as Hamlet, as well as more modern stories, like Star Wars.

Another notable publication is *Wide Ruled: A Friendly Interface to Author-Goal Based Story Generation* [26]. This tool provides an interface for authors to set up a number of story factors. The tool then builds and adjusts the story based off author input. We draw inspiration from this mixed initiative tool because it demonstrates how authors can systematically fill out details of a story or dialogue that the author may had forgotten about or did not plan for. At times, our tool does not provide all the necessary information that an author requires for their story, so implementing some of the features described in this paper may be useful for future versions of our tool.

The last area of relevant research draws inspiration from crowd sourcing knowledge for story generation. One pure mixed initiative approach allows an author to collaborate on a story with a computer by taking turns writing lines of text [1]. The system, developed by Say, is able to generate lines of text by mining Internet blog posts and selecting lines that appear relevant to the author's input. Although their system's output lacked coherence, another technique presented in [16] combines hand written stories in order to maintain the original creativity of the authors. The system outsources to workers from Amazon Mechanical Turk to write short stories about a given situation with simple language. After the system reads the story, a plot graph is built. These plot graphs are then combined in order to blend stories together. In a similar domain, our system crowd sources its knowledge by outsourcing to DBPedia.

3 IMPLEMENTATION

The implementation of our tool is composed of three modules. The first module is the Sparql back end, which serves as our connection to the DBPedia knowledge base. The second module is the tokenizer, which parses a text file containing grammar rules written by the author. The last module is the interpreter, which reads the tokenized grammar rules, prompts the user to provide the names of an arbitrary person and place, queries the Sparql backend to get the context, prompts the user to provide desired levels of rules expansions, and generates the story.

3.1 Sparql Back End

DBPedia altogether consists of 23 billion pieces of information (RDF triples) out of which 1.7 billion pieces of information are extracted from the English edition of Wikipedia. Our Sparql queries specifically allow authors to extract information about people and places that have corresponding Wikipedia web pages. Figure 1 shows an example Sparql query to extract an individual's name and a URI to the individual's spouse.

Much like SQL, Sparql begins with a SELECT statement, followed by a list of variables (each preceded by a question mark) that are to be used as the column values in the resulting table. There is no need for a FROM clause because our tool uses an endpoint located at http://dbpedia.org/sparql that is directly hooked

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?spouse WHERE {
  OPTIONAL {
    res:Barack_Obama foaf:name ?name.
  }
  OPTIONAL {
    res:Barack_Obama dbo:spouse ?spouse.
  }
}
```

Figure 1: A Sparql query that will fetch Barack Obama's name and a URI pointing to his spouse from the DBPedia database. The endpoint is located at http://dbpedia.org/sparql.

into the DBPedia database. Inside the WHERE clause, we specify the data that is to be extracted. The OPTIONAL keyword informs the query that a particular piece of information may not exist. Inside the OPTIONAL clause, there are three pieces of information that each correspond to the format of a RDF triple: subject, predicate, and object. In order to specify the location of a RDF triple inside a database, each piece of information must be preceded by a URI. For example, The resource object for Barack Obama is located at http://dbpedia.org/resource/Barack_Obama and the location for the name and spouse predicates are located at http://xmlns.com/foaf/0.1/name and http://dbpedia.org/ontology/spouse respectively. The last element in the RDF triple is the information that our query is to extract from the database. This information is then stored in the variables called "name" and "spouse", which match the variable names listed after the SELECT statement.

Although it is very likely that every person has a name[5], not every person has a spouse. For this reason, there is a chance that the table returned by the Sparql query will not contain some portion of the expected information. When designing grammars for our tool, authors must accommodate for missing pieces of information and should not rely too heavily on the existence of any individual data point. In order to maximize the probability of extrapolating additional information for people and places, two broad queries are written for demonstration purposes. The first retrieves person-related information and the second retrieves location-related information. The types of information that our Sparql back end can extrapolate from people and places are located in Tables 1 and 2.

3.2 Tokenizer

The second component of the tool is the tokenizer, which takes a text file containing grammar rules as input. The tokenizer uses regular expressions to pattern match the text file, which are then used to build an internal representation of the rules. The input grammar is structured similar to Backus-Naur style context free grammar rules[4][19]. Rules are formatted as, Key ::= Value, where keys on the left hand side are expanded into the values on the

[5]The "name" field is guaranteed to exist in our tool because a person's name is needed in order to execute the query.

Table 1: Potential person properties that can be pulled from DBPedia.

Key	Meaning
personName	The person's English name
birthPlace	City where the person was born
birthDate	The person's numerical birth date
description	A short description of the person's career
school	Where the person attended university
award	Any awards the person has received
religion	The person's active religion
residence	Where the person currently lives
spouse	Who the person is married to
children	A list of the person's children's names
parents	A list of the person's parent's names
hypernym	The person's profession
sex	Male, female, or other
networth	The person's net worth in scientific notation
fieldOfStudy	The person's field of study
knownFor	A short description of the person
nationality	The person's nationality

Table 2: Potential city properties that can be pulled from DBPedia.

Key	Meaning
cityName	The city's English name
country	The country where the city is located
nickname	Any nicknames that the city goes by
isPartOf	Description of the city
leaderName	Current political leader(s)
leaderTitle	Title(s) of current political leader(s)
populationTotal	Population count
east	Cities or land masses located east
north	Cities or land masses located north
northeast	Cities or land masses located northeast
northwest	Cities or land masses located northwest
south	Cities or land masses located south
southeast	Cities or land masses located southeast
southwest	Cities or land masses located southwest
west	Cities or land masses located west

right hand side. Values on the right hand side may be assembled in two different ways. First, they may simply be combined together to form one statement; this is the default case. Alternatively, they can be separated into sections using a series of bar operators in the following form: $Key ::= Value1|Value2$. The key has a random probability of expanding one of the sections on the right hand side and ignoring the other sections. We extend these basic definitions in order to give authors more control over rule expansions and streamline tedious story details.

Much like context free grammars, our grammar has both non-terminal and terminal nodes. However, our grammar makes an additional distinction between nodes that are guaranteed to exist and nodes that are not guaranteed to exist, which we coin as "optional" nodes. A node is guaranteed to exist if it is explicitly written by the author, while a node is optional if it's value has a dependency on a Sparql query. Nodes that are guaranteed to exist are surrounded by angle brackets and optional nodes are surrounded by square brackets. When defining the key for a non-optional node, the following form is used: $< Key > ::= Value$. When defining the key for an optional node, the user must specify the node's preconditions in parentheses following the key's name, thus the following form is used: $[Key](precondition) ::= Value$. Preconditions allow our tool to check whether or not values from a Sparql query exist prior to expanding the key. If the precondition for a key is not satisfied, then the key is ignored and is expanded to the empty string.

Figure 2 shows an example of a grammar with optional nodes to generate a sentence. Starting at the first line, the <Root> key will be expanded to two optional keys, [Name] and [Details], and one non-optional key, <Action>. The [Name] key is surrounded by square brackets because it has a dependency on personName from Table 1. If the personName value exists, then the key is expanded. Otherwise, the key is ignored. The [Details] optional key has a dependency on two values from Table 1, sex and description. As indicated by the \AND binary operator, if both of these values exist, then the key is expanded. Otherwise, if one of these values does not exist, the key is ignored. The <Action> key has no dependencies and will always be expanded to some text as well as the <Dream> key. The <Dream> key's value uses the \OVER binary operator, which is similar to the bar operator in that it splits the right hand side of the rule into multiple sections. However unlike the bar operator, the \OVER operator always expands the left key if the left key's precondition is satisfied. If the left key's precondition is not satisfied, then the \OVER operator will expand the right key. For example, the <Dream> key will expand the [Options] key if either birthPlace or school exists. If neither birthPlace nor school exists, then the <Dream> key expands the <Default> key. The \OVER operator is useful when the user would like to provide a default value in the case where an optional key's precondition is not satisfied.

Continuing with the example, the [Options] key has a dependency on one of two variables from Table 1, birthPlace or school. As indicated by the \OR binary operator, if both values are true, then [Options] will randomly expand to either [BirthPlace] or [School]. If only one of the values exists, then [Options] will expand to the key with the satisfied precondition. If neither of the values exist, then [Options] will expand to the empty string. Sample execution outputs of the grammar presented in Figure 2 can be found in Section 3.3.

In addition to optional nodes, our grammar includes a function called \CHOOSE. The \CHOOSE function gives the author some control over the length of the generated text. This is done by prompting the author to input a number indicating how many keys listed in the \CHOOSE function to expand. The first argument to the

```
<Root> ::= [Name][Details]<Action>
[Name](personName) ::= [personName]
[Details](sex \AND description) ::= ", the " [sex] " "
  [description]","
<Action> ::= " suddenly woke from a nightmare about their
  "<Dream> "."
<Dream> ::= [Options] \OVER <Default>
[Options](birthPlace \OR school) ::=
  [BirthPlace] | [School]
[BirthPlace](birthplace) ::= "troubling childhood in "
  [birthPlace]
[School](school) ::= "thesis defense at " [school]
<Default> ::= "secret affair"
```

Figure 2: A sample grammar that demonstrates the use of optional nodes, the \OVER operator, and how Sparql values can be used in grammar rules.

```
[Root](cityName) ::= "To explore the areas around "
  [cityName] ", our character " <Travel> "went to bed."
<Travel> ::= \CHOOSE("Travel", <Default>, [North],
  [South], [East], [West])
<Default> ::= "looked at a map, then "
[North](north) ::= <Moved>" north to " [north]", then "
[South](south) ::= <Moved>" south to " [south]", then "
[East](east) ::= <Moved>" east to " [east]", then "
[West](west) ::= <Moved>" west to " [west]", then "
<Moved> ::= "traveled" | "ran" | "walked" | "biked"
  | "flew" | "took an Uber"
```

Figure 3: A sample grammar that demonstrates the use of the \CHOOSE function. With the \CHOOSE function, an author can have some control over length of the generated text at runtime.

\CHOOSE function is a string description of the subsequent keys. The subsequent arguments are keys that can be expanded by the author. During runtime, the \CHOOSE function prompts the author to select a number ranging from zero to the number of provided keys with satisfiable preconditions. Given the author's input, the function then randomly selects the specified number of keys to expand. Figure 3 demonstrates the use of a simple \CHOOSE statement for a grammar that outputs text about a character traveling to different destinations. It is best practice to include at least one key that does not require any preconditions as a default value. Sample execution outputs of the grammar presented in Figure 3 can be found in Section 3.4.

3.3 Interpreter

The author directly interacts with our software using our interpreter. It is invoked with a Python call and an argument containing the location of the text file with the desired grammar. First, the interpreter will ask the author to enter a person, city, and state/province. The state/province may be needed alongside the city in order to provide a way to disambiguate places with similar names, like San Jose, California and San José, Costa Rica. The details are then used to build Sparql queries in order to retrieve relevant data from DB-Pedia. If no information is found on either the person or the city, the author will be prompted to input another person or city. This can occur when the author has made a spelling mistake or when the desired person or city does not have a Wikipedia page.

Once the interpreter has extracted all the relevant contextual details, it will forward the story grammar to the tokenizer to be broken into a list of statements that can be easily parsed. Once the grammar has been digested into a list of statements, the interpreter recursively expands each key until only strings and functions are left. When resolving a key's value, if the interpreter comes across a bar operator, only a single value from the list is used to resolve the key. We made a specific design decision to resolve each key independently every time it is used, rather than storing and reusing the result of the first expansion. Previous implementations where the value of each key was stored in memory resulted in text that always repeated in structure and content.

If the interpreter comes across a \CHOOSE function, it will first resolve all the keys in the parameter list, keeping only the keys with satisfied preconditions. Once the list of keys has been resolved, the author is then presented with a numerical choice between zero and the number of remaining keys. Given the author's numerical input x, the output text is formed by concatenating x random keys together. A high level overview of how the interpreter works is summarized in Figure 4.

3.4 Examples Executions

When the grammar from Figure 2 is sent to the interpreter, the program prompts the user to input a person's name in order to build and execute a Sparql query and fill out the information found in Table 1. If the author inputs the name "Napoleon", the Sparql backend returns the following set of keys with values: personName, birthPlace, birthDate, description, spouse, and sex. After tokenizing the grammar found in Figure 2, the interpreter may[6] print the following story:

"Napoleon, the male French monarch, military and political leader, suddenly woke from a nightmare about his troubling childhood in Corsica."

The grammar from Figure 2 is also adaptable to other author inputs. Say the author instead inputs the name "Barack Obama." Then the Sparql query connects with DBPedia and fills the following set of keys with values: personName, birthDate, description, birthPlace, school, award, religion, residence, spouse, children, parents, and sex. The resulting text after expanding the grammar could[7] be:

"Barack Obama, the male 44th President of the United States, suddenly woke from a nightmare about his thesis defense at Harvard Law School".

In our second example, the interpreter asks the author to input a name of a well known city. If the author inputs the city "San Jose", and then the state "California", the interpreter will send this information over to the Sparql backend. The result of the Sparql query is that the following keys are filled with values: cityName, country, isPartOf, leaderName, leaderTitle, populationTotal, east, north, northwest, south, southwest, and west. When the grammar from Figure 3 is processed by the tokenizer, the interpreter will resolve the story. The \CHOOSE function will trigger the interpreter to determine which optional nodes inside the \CHOOSE function resolve to valid strings. Since the preconditions for all the optional nodes are satisfied, there are five possible keys that the \CHOOSE function can expand. The interpreter then prompts the author to input a number between zero and five to randomly determine how many keys to expand. If the author inputs two, then a possible output after expanding the grammar is:

"To explore the area around San Jose, our character biked north to Milpitas, California, then took an Uber east to Mount Hamilton, California, then went to bed."

Another possible output after expanding the grammar for an input of two is:

"To explore the area around San Jose, our character looked at a map, then flew south to Morgan Hill, California, then went to bed."

If our author needs something particularly short, or the character is feeling particularly lazy, the author can input zero, which will result in:

[6] The output text is variable because Sparql may populate keys with multiple values. When this happens, a random value for each key is selected.
[7] As with the previous example, this example's output text is also variable because Barack Obama has attended multiple schools.

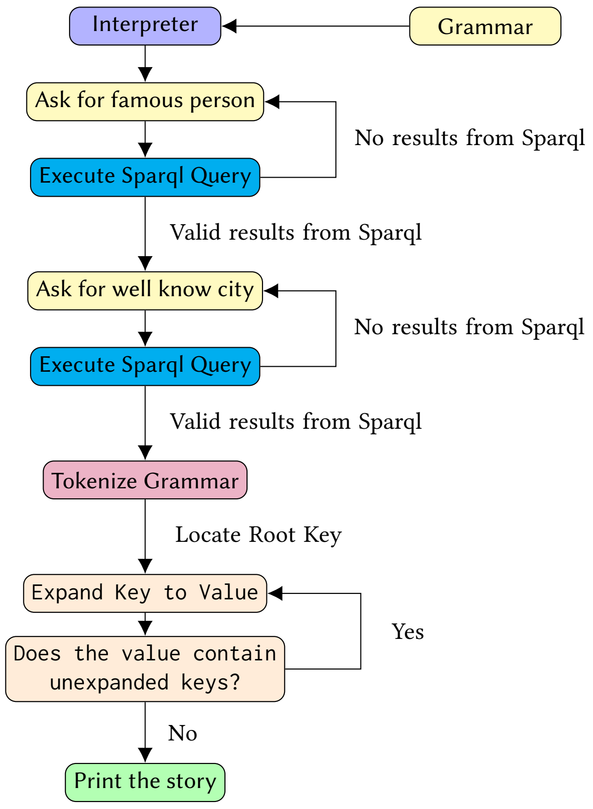

Figure 4: A flowchart representation of how a grammar and user inputs are combined by the interpreter in order to output text.

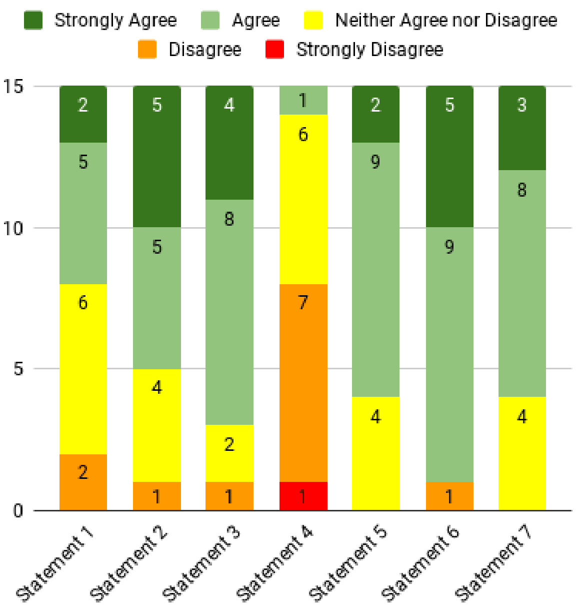

Figure 5: Results of the user study from 15 participants. The statements are located in Section 4

"To explore the area around San Jose, our character went to bed."

4 USER STUDY AND RESULTS

Quantifying the effectiveness of our tool is no trivial task. We designed a user study to test our hypothesis, that Semantic Web technologies can be used to generate more detailed text stories. Our study took the form of a survey that instructed the participants to use our tool and study our grammar rules. The survey begins with an informed consent agreement as well as an agreement that participants are at least 18 years old or older. Then, the following section of the survey collects information on the participants' levels of experience with computer science related topics. Additionally, all survey questions are designed to follow a standard five point Likert scale.

After demographic information is collected, the participants are instructed to experiment with three sample grammars. The first grammar is large and is intended to demonstrate how user inputs can influence character detail and story length. Participants are instructed to enter several different names, places, and different levels of desired detail for \CHOOSE functions. A sample run from the user study is shown in Appendix A.1. Once the user has generated two or three stories and has a high level understanding of how the tool works, the participants are directed to the Readme on our Github repository[8] to read about how our grammar is used to generate text. Having read the rules of the grammar, the participants are then instructed to study the grammars presented in Figures 2 and 3. Sample outputs from the user study are shown in Appendices A.2 and A.3 respectively. The purpose of this exercise is to determine how steep the learning curve is for understanding the syntax of our grammar and the usefulness of our grammar. Once the participants have completed the above tasks, the survey concludes by allowing the participants to rate their experience on a set of statements and short answer questions.

In total, 15 people completed the user study. From the demographic section of the survey, it can be concluded that the participants see themselves as moderately to very experienced computer scientists who have prior experience in using context free grammars. The participants are somewhat homogeneous because 11 out of 15 record that they are at least slightly experienced with procedural content generation for making text stories. Additionally, 8 out of the 15 participants record that they are at least slightly experienced with using SQL and most surprisingly, no participants have any experience with Sparql or any other RDF query language.

Once the participants finish experimenting with our tool and grammars, they were asked to rate their level of agreement with the following statements on a five point Likert Scale (strongly disagree, disagree, neither agree nor disagree, agree, and strongly agree):

(1) I felt in control of how much detail was included in the text.
(2) I felt that the person and location information enhanced the text.
(3) The tool generated details that were surprising and interesting that I did not expect.
(4) I felt that the generated details made sense in the context of the text.
(5) I see a positive application for Semantic Web technologies in future text generation research.
(6) I felt that the syntax of the grammar was easy to understand.
(7) I find that the grammar features extend the functionality of context free grammar text generators in a useful way.

Figure 5 shows the results of the survey. There were many positive responses to our tool and grammar. First of all, most of the participants agreed or strongly agreed that our tool generated details that were surprising and interesting that they did not expect. According to the free response section of the survey, participants especially enjoyed that our tool was able to extrapolate location information from arbitrary cities and character details that they did not know, such as aliases for a person's formal name. One participant expressed delight when the generated story referenced Mang0, the gamertag of Joseph Marques, whom the participant had entered, not expecting our grammar to know about or be able to reference gamertags. Second, participants agree that Semantic Web technologies can have a positive application for future text generation research. The participants further clarify that not only does the tool generate details that were surprising and interesting, but the participants also found that the generated text was relatable and funny because the tool utilized real world people and places in fictional settings. Third, most of the participants agreed or strongly agreed that our grammar's syntax was easy to read and the added features extended the functionality of context free grammar text generators in useful ways.

The overarching negative response about our tool and grammar was that the extrapolated details did not make complete sense within the context of the generated text. Participants clarified in

[8]https://github.com/ericlabouve/ProceduralStoryGeneration

the free response sections that some these errors can probably be removed with a more refined grammar. However, other errors are more difficult to handle. For example, the third sentence of the generated story in Appendix A.1 is grammatically incorrect: "Now, Barack Obama lives at White House." These types of grammatical errors are common because Wikipedia has inconsistent naming conventions. A possible method for fixing some of these issues would be to refine the output text with an English grammar parser, such as those provided by nltk[9] or Stanford[10].

Another issue that participants pointed out was that sometimes the system would extract outdated information. For example, there was one case where a participant input "Donald Trump" and the tool mentioned one of his ex wives instead of his current wife, Melania Trump. This output occurred because our tool selects a random value for each key that has multiple return values from a Sparql query. Upon further investigation, it appears that Sparql queries return values in chronological order. This structure can probably be utilized for keys containing time-dependant data. Lastly, participants also shared their thoughts on where our system can be improved. Many agreed that the information mined about people should be more relevant to the generated text so that story plots can be more geared toward user input. This would be challenging to accomplish with our tool because we rely on a pre-written grammar. Additionally, the availability of RDF triples mined from DBPedia may be too sparse for such an application. However with a more targeted RDF triple data sets, such as what might be maintained for the story coherence of a game, we can see foresee an implementation where the mined information influences the plot.

5 CONCLUSION

We have presented a text generation tool that takes advantage of the Semantic Web to extrapolate story details that are not explicitly written by the author. A user study was conducted to verify our hypothesis, which was whether or not the Semantic Web can be used to generate more in-depth text stories. Our user study had 15 participants interact with our tool and record their responses on Likert scales and free response sections. Overall, the participants expressed that the tool generated details that were surprising and interesting and that our grammar and functions were easy to understand and expanded the capabilities of context free grammars in useful ways. One thing we learned was that we should have made it clear to the test subjects that the generated stories are a fictionalized account of real people, so that non-factual statements would not be a surprise for them.

The results show evidence in our hypothesis, that Semantic Web technologies can be used to generate more in-depth text stories and participants see a positive application for Semantic Web technologies in story generation systems in the future.

6 FUTURE WORK

Sparql is a useful tool for extrapolating information for specific pieces of data. However, it is hard to identify general objects that may not have a dedicated Wikipedia page and DBPedia object. For example, there is no Wikipedia page for a "Panda," but there is a page for a "Giant Panda." The distinction between "Panda" and "Giant Panda" is not something someone would think about ahead of time when searching for pandas on the web. But these slight differences will result in runtime errors when the Sparql query searches for the appropriate DBPedia object. Future research would determine a technique for guessing the appropriate DBPedia object when provided a vague or misspelled user input. One avenue for implementation would be to plug the word into Google Search and then return the URL of the first Wikipedia page that Google Search returns. This URL can then be parsed to determine the URI of the appropriate DBPedia object.

Another area of future research would be query automation. The Sparql queries used to collect person and location information in this paper were hand crafted according to specific resources available in the DBPedia ontology. This process was very tedious, but we see room for automation. A tool that writes Sparql queries can be used to broaden the scope of topics available for user input would be useful for our application.

Our tool could also be expanded to use multiple people and places, or even recursive references. Clearly defining multiple people and places may be challenging in the grammar, but it would allow for stories with more than one main character and multiple locations that are not immediately connected to each other. This is often the case in stories and video games. Recursive references of characters could allow for more in depth references to people, like a character's spouse, children, parents, neighbors, coworkers, or many other relations, all without previous knowledge from the author.

[9]http://www.nltk.org/

[10]https://nlp.stanford.edu/software/lex-parser.shtml

REFERENCES

[1] [n. d.]. Say Anything: A Massively collaborative Open Domain Story Writing Companion. Erfurt, Germany.

[2] Barros Gabriella A.B., Liapis Antonios, and Togelius Julian. 2016. Playing with Data: Procedural Generation of Adventures from Open Data. In *DiGRA/FDG &#3916 - Proceedings of the First International Joint Conference of DiGRA and FDG*. Digital Games Research Association and Society for the Advancement of the Science of Digital Games, Dundee, Scotland.

[3] Isabel Alexandre, Ana Paiva, and Paul Brna. 2003. Real characters in virtual stories: Promoting interactive story-creation activities. (05 2003).

[4] John W Backus. 1959. The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. *Proceedings of the International Comference on Information Processing, 1959* (1959).

[5] Byung-Chull Bae and R. Michael Young. 2014. A Computational Model of Narrative Generation for Surprise Arousal. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 2 (June 2014), 131–143. https://doi.org/10.1109/TCIAIG.2013.2290330

[6] Eric Buckthal and Foaad Khosmood. 2014. (Re)telling Chess Stories as Game Content. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*.

[7] Vinh Bui, Hussein Abbbass, and Axel Bender. 2010. Evolving stories: Grammar evolution for automatic plot generation. In *IEEE Congress on Evolutionary Computation*. 1–8. https://doi.org/10.1109/CEC.2010.5585934

[8] Ronan Champagnat, Guylain Delmas, and Michel Augeraud. 2010. A storytelling model for educational games: Hero's interactive journey. *International Journal of Technology Enhanced Learning* 2 (01 2010). https://doi.org/10.1504/IJTEL.2010.031257

[9] Fred Charles, Julie Porteous, Marc Cavazza, and Jonathan Teutenberg. 2011. Timeline-based navigation for interactive narratives. 37. https://doi.org/10.1145/2071423.2071469

[10] Yun-Gyung Cheong, Kinam Park, Woo-Hyun Park, and Byung-Chull Bae. 2017. A Database-centric Architecture for Interactive Storytelling. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. Article 49, 4 pages.

[11] Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on information theory* 2, 3 (1956), 113–124.

[12] Richard E Fikes and Nils J Nilsson. 1994. STRIPS, a retrospective. *Artificial intelligence in perspective* 227 (1994).

[13] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. 1997. Extending planning graphs to an ADL subset. In *European Conference on Planning*. Springer, 273–285.

[14] Ben Kybartas and Rafael Bidarra. 2017. A Survey on Story Generation Techniques for Authoring Computational Narratives. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 3 (Sept 2017), 239–253. https://doi.org/10.1109/TCIAIG.2016.2546063

[15] Ben Kybartas and Clark Verbrugge. 2014. Analysis of ReGEN as a Graph-Rewriting System for Quest Generation. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 2 (June 2014), 228–242. https://doi.org/10.1109/TCIAIG.2013.2290088

[16] Boyang Li, Stephen Lee-Urban, and Mark Riedl. 2013. Crowdsourcing interactive fiction games. In *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14-17, 2013*. 431–432.

[17] Yoichi Matsuyama, Akihiro Saito, Shinya Fujie, and Tetsunori Kobayashi. 2015. Automatic Expressive Opinion Sentence Generation for Enjoyable Conversational Systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23, 2 (Feb 2015), 313–326. https://doi.org/10.1109/TASLP.2014.2363589

[18] Eric Miller. 1998. An Introduction to the Resource Description Framework. *Bulletin of the American Society for Information Science and Technology* 25, 1 (11 1998), 15–19. https://doi.org/10.1002/bult.105

[19] Peter Naur, John W Backus, Friedrich L Bauer, Julien Green, C Kafz, John McCarthy, Alan J Perlis, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, et al. 1997. Revised Report on the Algorithmic Language A lgol 60. In *ALGOL-like Languages*. Springer, 19–49.

[20] Lyn Pemberton. 1989. A Modular Approach to Story Generation. In *Proceedings of the Fourth Conference on European Chapter of the Association for Computational Linguistics (EACL ’89)*. 217–224.

[21] Vladimir Propp. 1968. *Morphology of the Folktale*. Univ. Texas Press.

[22] Aaron A. Reed, Jacob Garbe, Noah Wardrip-Fruin, and Michael Mateas. 2014. Ice-Bound: Combining Richly-Realized Story with Expressive Gameplay. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*.

[23] Owen Sacco, Antonios Liapis, and Georgios N. Yannakakis. 2016. A holistic approach for semantic-based game generation. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. 1–8. https://doi.org/10.1109/CIG.2016.7860386

[24] Stephen Scheinberg. 1960. Note on the Boolean properties of context free languages. *Information and Control* 3, 4 (1960), 372–375.

[25] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

[26] James Skorupski, Lakshmi Jayapalan, Sheena Marquez, and Michael Mateas. 2007. Wide Ruled: A Friendly Interface to Author-Goal Based Story Generation. In *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, Marc Cavazza and Stéphane Donikian (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 26–37.

[27] Rafael Pérez y Pérez and Mike Sharples. 2001. MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental and Theoretical Artificial Intelligence* 13 (04 2001), 119–139. https://doi.org/10.1080/09528130118867

A APPENDIX

The grammar that generates the story in Section A.1 can be found on our Github. The general plot of the story in Section A.1 is for a character to seek out a lost diamond, but he/she is stopped multiple times by enemies. Please note that we use the gender neutral pronoun “they" instead of “he" or “she" because our grammar does not disambiguate the person's sex. Also note that some parts of the story are grammatically incorrect because some values in DBPedia have inconsistent formatting.

A.1 Story Grammar

```
$ python3 proceduralStoryGeneration.py storyGrammar.txt
Please enter a famous person's name: Obama
Retrieving information, one moment...
No info was found on Obama, please chose another
Please enter a famous person's name: Barack Obama
Retrieving information, one moment...
Please enter a well known city: Manila
Is this city located in America?
If so, please enter the state/province. Otherwise, please press enter:
Retrieving information, one moment...
Finished getting context. Assembling story...
How much detail about Character Detail ? Please enter a number between 0 and 7
7
How much detail about Travel Extent? Please enter a number between 0 and 8
4
The generated story:
There once was a Person named Barack Obama. Barack Obama was born in Kapiolani Medical Center for Women and Children on 1961-08-04
to Ann Dunham. Now, Barack Obama lives at White House. They were raised to believe in Protestantism. Later in life, Barack Obama became
a 44th President of the United States. Throughout Barack Obama's successful career, they received numerous awards such as Nobel Peace
Prize. When Barack Obama came of age, they studied at Occidental College. Barack Obama, tired of their regular life, traveled to Manila
in order to come across the breathtaking diamond. However, smelly robots invade Barack Obama! Then Barack Obama defeats them all.
Afterwards, Barack Obama crawls over to a local shop owner and asks, 'Is the diamond in this city?' They respond, 'Legend says its gone!'
To continue the hunt, Barack Obama walks to a small city outside town. Then, warlike dogs mug the civilians! Then Barack Obama runs
them out of town. Afterwards, Barack Obama crawls over to a defeated enemy and asks, 'Where can I find the diamond !' They respond,
'Oh! Umm... No I don't recall ever seeing such a thing.' To continue the quest, Barack Obama flies northwest to Navotas. All of a sudden,
quarrelsome samurai rush Barack Obama ... Then Barack Obama barely wins the fight. Afterwards, Barack Obama approaches a defeated
enemy and asks, 'Is the diamond in this city?' They respond, 'I wouldn't tell you even if I knew!' To continue the quest, Barack Obama takes
an Uber south to Pasay. Suddenly, aggressive vampires intrude the civilians ... Then Barack Obama hides until everyone leaves. Afterwards,
Barack Obama crawls over to a a defeated enemy and asks, 'Is the diamond in this city?' They respond, 'Oh! Umm... No I don't recall ever
seeing such a thing.' To continue the adventure, Barack Obama bikes southeast to Makati. However, giant dogs strike the buildings. Then
Barack Obama wins the battle. Afterwards, Barack Obama approaches a wounded civilian and asks, 'Is the diamond in this city?' They
respond, 'How would I know? Try somewhere else?' When finding the diamond seemed hopeless, Barack Obama sits on the ground and
Barack Obama finally sees the diamond in the distance.
```

A.2 Sample Grammar 1

```
$ python3 proceduralStoryGeneration.py sampleGrammar2.txt
Please enter a famous person's name: Will Smith
Retrieving information, one moment...
Please enter a well known city: San Francisco
Is this city located in America?
If so, please enter the state/province. Otherwise, please press enter: California
Retrieving information, one moment...
Finished getting context. Assembling story...
The generated story:
Will Smith, the male American actor, film producer and rapper, suddenly woke from a nightmare about his troubling childhood in Philadelphia.
```

A.3 Sample Grammar 2

```
$ python3 proceduralStoryGeneration.py sampleGrammar1.txt
Please enter a famous person’s name: Will Smith
Retrieving information, one moment...
Please enter a well known city: San Francisco
Is this city located in America?
If so, please enter the state/province. Otherwise, please press enter: California
Retrieving information, one moment...
Finished getting context. Assembling story...
How much detail about Travel? Please enter a number between 0 and 5
2
The generated story:
To explore the areas around San Francisco, our character took an Uber east to Alameda County, then flew north to Sausalito, California, then
went to bed.
```